Введение в Docker

Что такое Docker и для чего он нужен. Запуск контейнеров, монтирование в них папок хостовой системы, проброс портов наружу.

[Введение](#_x2bkhx2fztof)

[Установка Docker](#_ymnshjogoz3y)

[Домашнее задание](#_qk61dk4of60m)

[Дополнительные материалы](#_y937sk8fclye)

[Используемая литература](#_uvp6qax5r1ok)

# Введение

Docker – программное обеспечение для автоматизации развертывания и управления приложениями в среде виртуализации на уровне операционной системы. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесен на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами. Изначально он использовал возможности LXC, с 2015 года применял собственную библиотеку для контейнерной виртуализации – libcontainer. С появлением Open Container Initiative начался переход от монолитной к модульной архитектуре.

Из чего же состоит Docker и что он использует?

**Image (образ)** – в нем сохраняется собранная подсистема, необходимая для работы процесса.

**Container (контейнер)** – процесс, инициализированный на базе образа. То есть контейнер существует, только когда запущен процесс.

**Host (хост)** – среда, в которой запускается Docker, сервер или виртуальная машина (последний вариант, конечно, – не лучшая идея).

**Volume** – это дисковое пространство между хостом и контейнером. Проще говоря, это папка на сервере, примонтированная внутрь контейнера.

**Dockerfile** – файл с набором инструкций для создания образа будущего контейнера.

**Service (сервис)** – по сути, это запущенный образ (один или несколько контейнеров), дополнительно сконфигурированный такими опциями, как открытие портов, маппинг папок (volume) и прочее. Обычно это делается при помощи файла docker-compose.yml.

**Docker hub** – открытое хранилище образов Docker, как GitHub для кода. Имеет второе название – registry. Если у вас инфраструктура в Docker, нужно иметь свое registry с образами для ваших проектов, чтобы разворачивать контейнеры оттуда.

**Docker-compose** (докер-композ, чаще композер, но не путать с php composer) – утилита, облегчающая сборку и запуск системы, состоящей из нескольких контейнеров, связанных между собой. Это простой вариант оркестрации. Если нужно что-то посерьезней, используйте Kubernetes.

**Build (билд)** – процесс создания образа из набора инструкций в докерфайле или нескольких докерфайлах, если билд делается с помощью докер-композера.

# Установка Docker

Начнем с установки. Можно поставить Docker из репозиториев, которые есть в Ubuntu, но там довольно старые версии и лучше подключить репозиторий Docker. Все это описано на сайте с документацией для разных операционных систем.

Добавляем репозиторий:

|  |
| --- |
| apt updateapt install apt-transport-https ca-certificates curl software-properties-commoncurl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" |

Устанавливаем Docker. Обратите внимание, что нужен именно docker-ce. Это связано с ребрендингом и всякими маркетинговыми играми компании.

|  |
| --- |
| apt updateapt install docker-ce |
|  |

Запустим что-нибудь простое. Идем на docker hub, ищем официальный debian-контейнер (<https://hub.docker.com/_/debian/>) и выполняем команду из консоли root или используя sudo (для всех остальных команд такая же схема):

|  |
| --- |
| docker run debian:latest echo "test run" |

Как видите, команда отработала в контейнере, и на этом он выключился, но не удалился: сами контейнеры не удаляются, это надо делать вручную. Команда docker ps покажет все запущенные контейнеры, docker ps -a дополнительно покажет выключенные. Для удаления используем docker rm имя\_контейнера, например:

|  |
| --- |
| docker rm objective\_mirzakhani |

Конечно, так контейнеры никто не использует – в них запускают процессы, например nginx. Для его запуска используем команду:

|  |
| --- |
| docker run --name test-nginx -p 8080:80 -d nginx:1.12-alpine |



Здесь добавлено имя контейнера и проброс портов делается при помощи флага -p: слева указывается порт на сервере, справа порт контейнера. Для реализации этого функционала docker использует NAT-таблицу iptables. Флаг -d позволяет запустить контейнер с отсоединенным терминалом. Это чем-то похоже на использование & в конце команд в Linux.

Проброса портов мало, и почти всегда надо примонтирвать файлы или папки внутрь контейнера, чтобы прокинуть конфиг в контейнер или, наоборот, вытащить из него файлы базы данных, чтобы они не удалились вместе с контейнером. Тут используется флаг -v. Запустим контейнер postgreSQL, чтобы проверить это:

|  |
| --- |
| docker run --name test-postgres -p 5432:5432 -v /tmp/postgres:/var/lib/postgresql/data -d postgres:9.6 |



Владелец и группа у файлов и папок имеют права 999. Что это такое и откуда оно взялось? Заглянем внутрь контейнера, чтобы это понять:

|  |
| --- |
| docker exec -it test-postgres bashls -la /var/lib/postgresql/datacat /etc/passwd | grep postgres |

Теперь должно быть понятно, откуда взялись 999, а если нет, освежите знания по linux. Как вы уже поняли, docker exec позволяет зайти внутрь контейнера, если точнее, выполнить команду внутри него. Флаги -it – это интерактивный терминал. Для выхода из контейнера служит команда exit. Команда может быть любой: можно сразу зайти в консоль postgreSQL, затем выйти оттуда и т. д.:

|  |
| --- |
| docker exec -it test-postgres psql -U postgres\qdocker exec -it test-postgres mkdir /tmp/new\_folderdocker exec -it test-postgres ls /tmp |

# Домашнее задание

1. Установить Docker, создать контейнеры для nginx и perconadb/postgresql (на выбор).
2. Собрать собственный контейнер php-fpm на основе имеющегося c DockerHub.

# Дополнительные материалы

1. <https://docker.com>
2. [https://hub.docker.com/\_/debian](https://hub.docker.com/_/debian/)

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [https://hub.docker.com/\_/debian](https://hub.docker.com/_/debian/)